Lab 6
November 6, 2017
Digital Filter Design

INSTRUCTIONS:
All lab submissions include a written report and source code in the form of an m-file. The report contains all plots, images, and figures specified within the lab. All figures should be labeled appropriately. Answers to questions given in the lab document should be answered in the written report. ***The written report must be in PDF format***. Submissions are done electronically through Compass2G.

1 Ideal Filters

A filter $H_d(\omega)$ has can be decomposed into magnitude and phase components,i.e. $H_d(\omega) = |H_d(\omega)|e^{i\angle H_d(\omega)}$. For an ideal filter, $\angle H_d(\omega) = 0$. That is, there is no phase shift associated with any ω. The most 3 basic types of filters are highpass, lowpass, and bandpass filters. The magnitude response of each of these filters are:

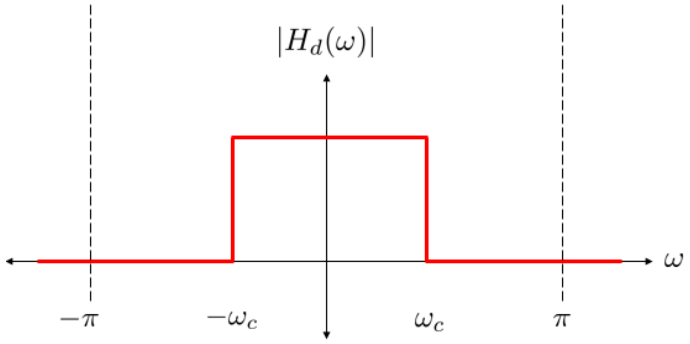


Figure 1: Ideal LPF magnitude response.

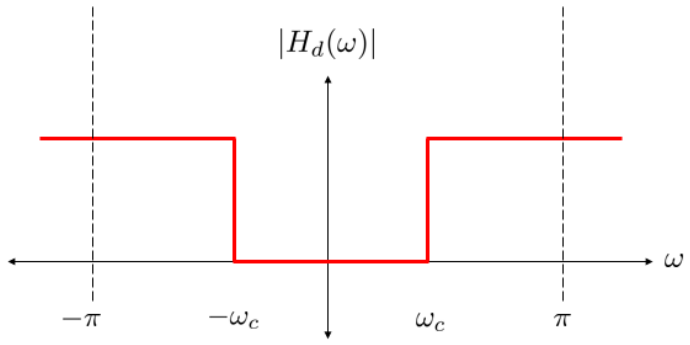


Figure 2: Ideal HPF magnitude response.

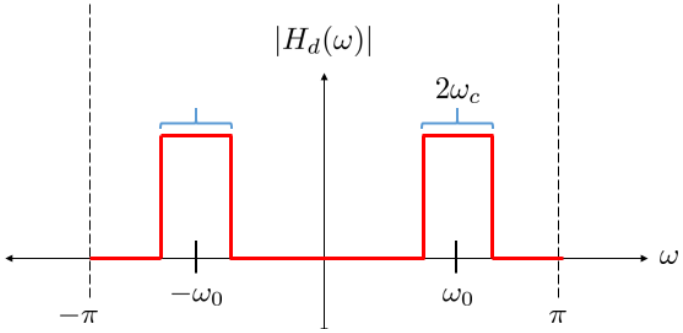


Figure 3: Ideal BPF magnitude response.

where the corresponding impulse response for each is

$$h(n) = \frac{\omega_c}{\pi}\text{sinc}\left(\frac{\omega_c}{\pi}n\right) \qquad \text{(Lowpass)}$$

$$h(n) = \delta(n) - \frac{\omega_c}{\pi}\text{sinc}\left(\frac{\omega_c}{\pi}n\right) \qquad \text{(Highpass)}$$

$$h(n) = \cos(\omega_0 n)\frac{\omega_c}{\pi}\text{sinc}\left(\frac{\omega_c}{\pi}n\right) \qquad \text{(Bandpass)}$$

and the definition of sinc is the same as MATLAB's.

Report Item 1: Implement the impulse response for the ideal lowpass, highpass, and bandpass filters. Let $n = -N : N$ for $N = 20$ and choose ω_c and ω_0 such that the magnitude response "makes sense". Plot each impulse response using **stem** and the magnitude response using **plot**. Repeat for $N = 40$. How does the magnitude response differ from the ideal and why?

In general, an ideal filter cannot be realized due to the fact that it is infinite in length and non-causal. A remedy is to first perform a shift followed by a truncation. This makes the filter both finite and causal at the cost of adding ripples to the magnitude response and linear phase to the phase response. The formal name for these ripples is *Gibb's Phenomenon*. Truncation also affects the transition width of the filter. The transition width is the bandwidth between the passband and stopband edge frequencies. In general, the longer the filter is, the smaller the transition width.

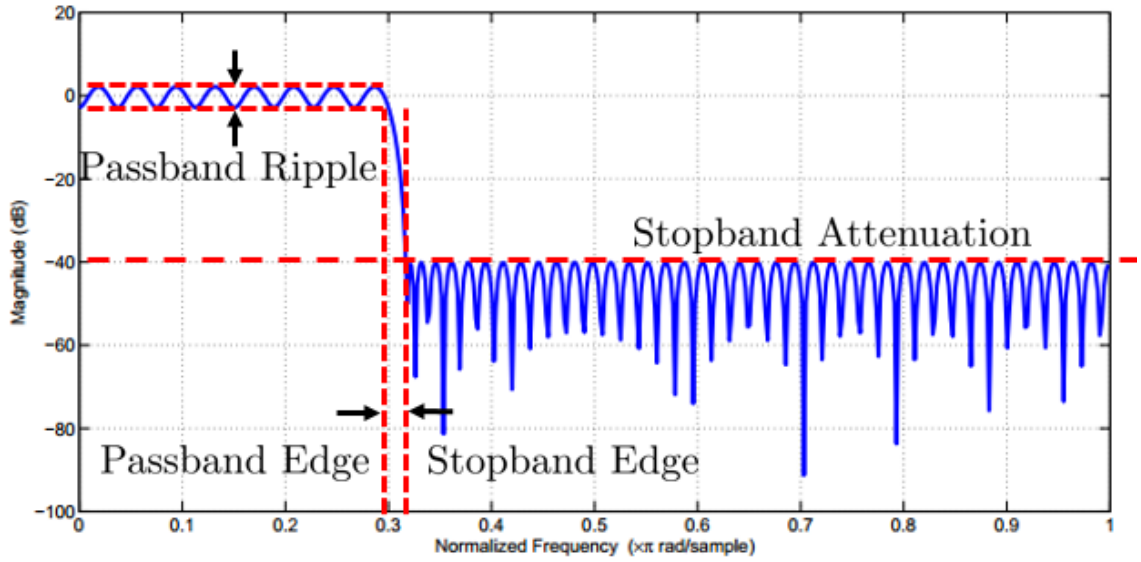


Figure 4: Filter metrics.

Passband ripple is the peak-to-peak ripple in the passband in dB. Stopband attenuation is the highest peak in the stopband. The transition width is the difference between the stopband and passband edges. The location for the stopband and passband edges are arbitrary and depend on the convention used.

Report Item 2: Load *impulseresponse.mat* which contains the impulse response of a filter h. Use **stem** to plot the impulse response. Find the magnitude response (in dB) and phase response of this filter and **plot** both. For visualization purposes, you may want to consider zero-padding your signal to 512 length. What are the stopband and passband ripples? What is the transition bandwidth?

2 Generalized Linear Phase FIR Filters

Filters can be separated into two classes: finite impulse response (FIR) and infinite impulse response (IIR). As implied by their names, FIR filters have a finite length impulse response and IIR filters have an infinite length impulse response. The transfer function for an FIR filter can be expressed as a polynomial of the form:

$$y(n) = b_0 x(n) + b_1 x(n-1) + \cdots + b_{N-1}(x - N + 1) \tag{1}$$

$$H(z) = b_0 + b_1 z^{-1} + \cdots + b_{N-1} z^{-(N-1)} \tag{2}$$

where the output only depends on the input. Using the discrete-time delay property of z-transforms, the transfer function for FIR filters can easily be determined. FIR filters are inherently stable because they have no poles (except the ones at zero). A symmetric FIR filter always has linear phase. FIR filters can be designed using several methods. Some common approaches are the windowing method, frequency sampling method, and Parks-McClellan method.

Consider a filter with a real-valued, length N impulse response $\{h_n\}_{n=0}^{N-1}$ which has a DTFT of the form

$$H_d(\omega) = R(\omega) e^{i(\alpha - \omega M)} \tag{3}$$

where $R(\omega)$ is real-valued, α is a real-valued constant, and $M = \frac{N-1}{2}$. Note that

$$R(\omega) = \begin{cases} |R(\omega)| & \text{if } R(\omega) > 0 \\ e^{\pm j\pi}|R(\omega)| & \text{if } R(\omega) < 0. \end{cases}$$

Thus, a sign change in $R(\omega)$ results in a $\pm\pi$ jump in the phase of $H_d(\omega)$. Phase jumps of π may not be 'unwrapped' like jumps of 2π, but the filter $H_d(\omega)$ will still have generalized linear phase.

2.0.1 Type I Generalized Linear Phase

An FIR filter with real-valued impulse response $\{h_n\}_{n=0}^{N-1}$ as specified above has type I generalized linear phase if and only if it has even symmetry given by

$$h[n] = h[N-1-n] = \begin{cases} n = 0, 1, \ldots, N/2-1 & N \text{ even} \\ n = 0, 1, \ldots, (N-1)/2 & N \text{ odd.} \end{cases} \tag{4}$$

Considering Eq. 3, it may be proved that for type I generalized linear phase $\alpha = 0$ and $R(\omega)$ must be an even function (e.g. $R(\omega) = R(-\omega)$, a sum of cosine functions).

An example of type I generalized linear phase for an FIR filter is shown in Fig. 5.

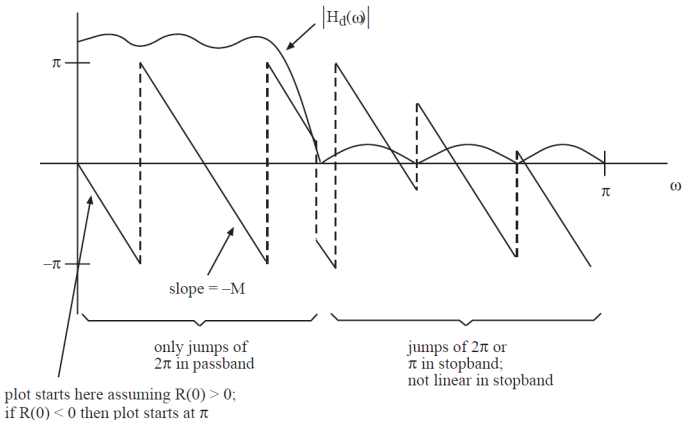


Figure 5

2.0.2 Type II Generalized Linear Phase

An FIR filter with real-valued impulse response $\{h_n\}_{n=0}^{N-1}$ has type II generalized linear phase if and only if it has odd symmetry given by

$$h[n] = -h[N-1-n] = \begin{cases} n = 0, 1, \ldots, N/2-1 & N \text{ even} \\ n = 0, 1, \ldots, (N-1)/2 & N \text{ odd} \end{cases} \tag{5}$$

Considering Eq. 3, it may be proved that for type II generalized linear phase $\alpha = \pm\frac{\pi}{2}$ and $R(\omega)$ must be an odd function (e.g. $R(\omega) = -R(-\omega)$, a sum of sine functions). If N is odd, $h[M]$ $(M = (N-1)/2)$ must be zero to satisfy the odd symmetry condition.

3 Windowing Method

Let us begin with the ideal lowpass filter, shown below (left), with the desired frequency response $D(\omega)$. We might choose the filter coefficients to be the inverse DTFT of the ideal lowpass filter, $d[n] = \frac{\omega_c}{\pi}\text{sinc}[\omega_c n]$, as shown below (right).

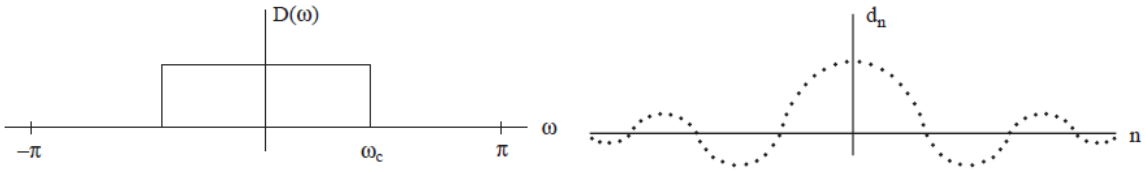


Figure 6

However, the sequence $d[n]$ is infinitely long and non-causal. To obtain an FIR filter, it is necessary to shift the sequence $d[n]$ and truncate it in time. The shift of the impulse response adds linear phase to the complex frequency response. The truncation causes sidelobes in the magnitude response, known as Gibb's phenomenon. These sidelobes can be reduced by windowing the impulse response (in the time domain), at the cost of increasing the transition bandwidth in the magnitude frequency response.

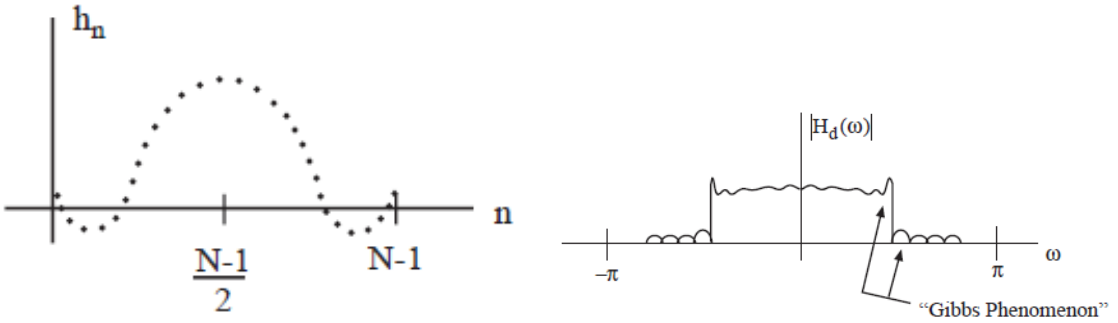


Figure 7

To design a generalized linear phase filter of coefficients $\{h_n\}_{n=0}^{N-1}$ such that $|H_d(\omega)|$ approximates the desired frequency response:

1. Define ω as you would for FFT.
2. Let $G_d(\omega) = D(\omega)e^{-jM\omega}$, where $M = \frac{N-1}{2}$ and $D(\omega) = 1$ for $|\omega| < \omega_c$.
3. Find $g[n] = \text{DFT}^{-1}\{G_d(\omega)\}$
4. Let $h[n] = w[n]g[n]$, where $w[n]$ is the window function (e.g. Hamming window centered at $n = M$)

Report Item 3: Design and analyze the following generalized linear phase FIR filters using the windowing method. Design a lowpass filter of length $N = 25$ with cutoff frequency $\omega_c = \pi/3$ using a Hamming window. First derive an expression for $h[n]$ (it may contain summations and an inverse DFT) and apply the required window, then calculate and plot the impulse response, and the magnitude (in dB) and phase of the frequency response of the designed FIR filter. Find the passband ripple, stopband attenuation, passband edge frequency, and stopband edge frequency.

4 Frequency Sampling Method

As the name implies, this design method is based on sampling the ideal frequency response. Let $G_d(\omega)$ be the ideal frequency response, with linear phase. We then force $H_d(\omega)$ to agree with $G_d(\omega)$ at frequency-sampled points $\omega = \frac{2\pi m}{N}$ for $0 \leq m \leq N-1$. The impulse response is determined by the inverse DFT

$$h[n] = \frac{1}{N}\sum_{m=0}^{N-1} G_d(\tfrac{2\pi m}{N}) e^{j\frac{2\pi mn}{N}} \quad 0 \leq n \leq N-1.$$

The ideal and frequency-sampled filters are shown in Fig. 8.

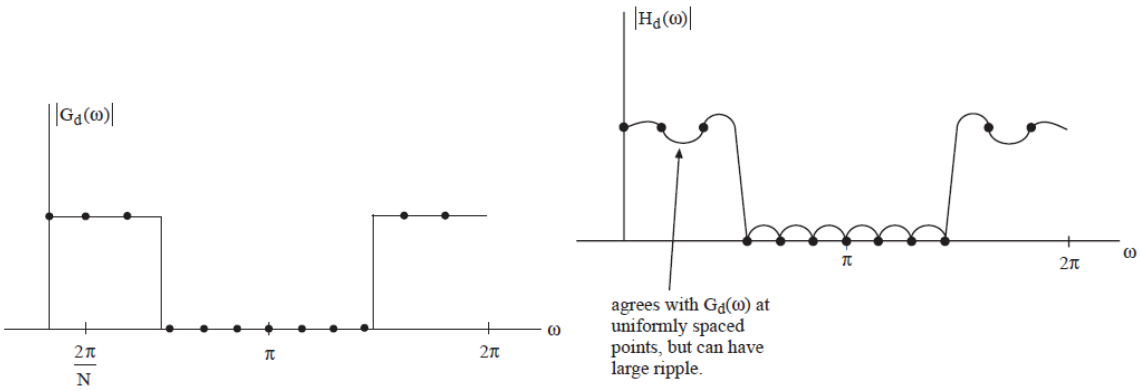


Figure 8

Note that the two frequency responses $H_d(\omega)$ and $G_d(\omega)$ are only identical for the sampled values. The implementable frequency response $H_d(\omega)$ can have significant ripples. One way to reduce the ripples is to allow for a variable transition sample in the transition band, whose amplitude can be chosen to utilize tradeoffs between ripple amplitudes and the transition bandwidth. Such a sample is represented with an 'x' in the figure below.

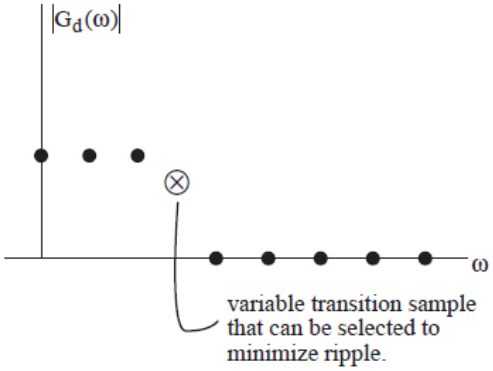


Figure 9

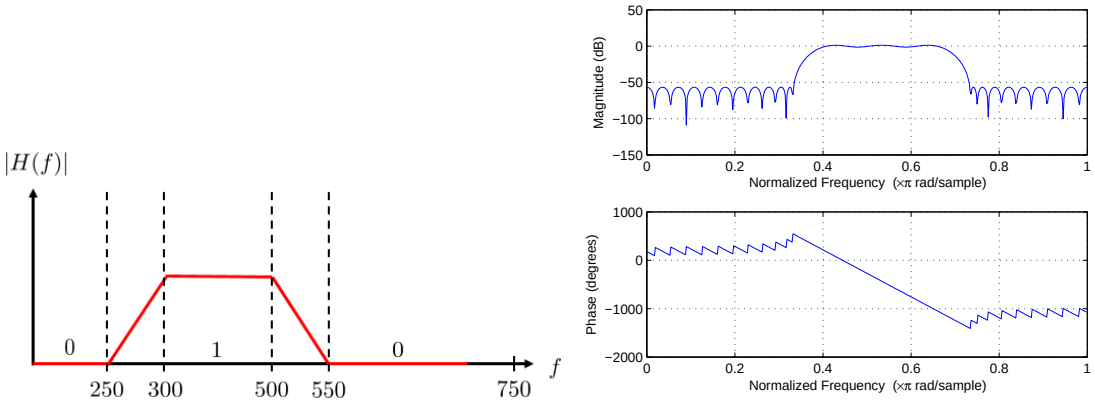
5 Parks-McClellan FIR Filter Design

The Parks-McClellan FIR filter produces a minimum length, linear phase filter with minimal error between the desired response and the actual response using the Remez exchange algorithm. In MATLAB, a PM filter can be designed using **firpmord**. The input to this command are the band edges, desired amplitude, and ripple levels. A bandpass filter can be specified by **firpm**(f, a, rp, fs) where $f = [250, 300, 500, 550]$, $a = [0, 1, 0]$, $rp = [0.001, 0.1, 0.001]$, and $f_s = 1500$. This corresponds to an ideal bandpass filter shown in Figure 11a. The output of **firpmord** are the filter order, frequency band edges, desired amplitudes, and weighting function. These are given to **firpm** which returns the FIR filter coefficients b. The filter returned from **firpm** can be applied to an input signal using **filter**.

```
clc, clear all, close all

f = [250,300,500,550];
a = [0,1,0];
rp = [0.001 0.1 0.001];
fs = 1500;
[n,fo,mo,w] = firpmord(f, a, rp, fs);
b = firpm(n,fo,mo,w);
freqz(b,1);
```

Figure 10: Using firpm to produce a bandpass filter using **firpm**.



(a) Desired response. (b) Actual response.

Figure 11: A comparison between the desired response vs the actual response produced by the PM filter method.

Report Item 4: Design a low-pass filter using the PM filter method with a passband edge frequency of 0.3π and a stopband edge frequency of 0.36π with a stopband attenuation of -50 dB and passband ripple of 2 dB peak-to-peak. Use **freqz** to plot the magnitude and phase response of this filter. Use **impz** to plot the impulse response of this filter.

6 Fourier De-noising

In this section, we apply our knowledge about filters to remove noise from a signal. But first of all, what is noise? Well in the case of audio, it can be considered sound which our ears find unpleasant. Mathematically there is little difference between signal and noise and sometimes it can be hard to separate the two from each other. A particular kind of noise known as *white noise* is broadband in frequency. That is, if you take the DFT of white noise, the frequency spectrum has similar amplitude for all frequencies and random phase.

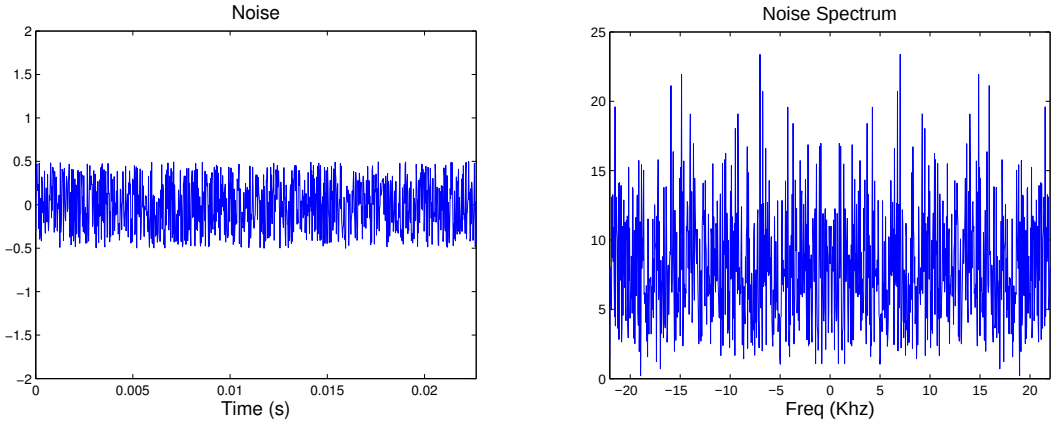


Figure 12: White noise.

Report Item 5: Load *sound1.wav* using **wavread**. How many seconds is the total sound file? Plot the magnitude spectrum along with the spectrogram of the signal. Write the parameters you choose for the spectrogram. Use **firpm** to create a filter that will remove the noise observed in the frequency response. Plot the magnitude spectrum and spectrogram of the filtered signal using the same parameters as before. Additionally, use **soundsc** to listen to the filtered signal. Comment on the effectiveness of the filter.

Using filter techniques, it is possible to remove noise within certain frequency bands. However, there are limitations to this technique. If the noise is truly broadband, then noise components can overlap with the frequency components of the actual signal. It is difficult to remove this kind of noise using Fourier de-noising techniques. An alternative method for de-noising is wavelets.

Report Item 6: Load *sound2.wav* using **wavread**. Plot the magnitude spectrum along with the spectrogram of the signal. Write the parameters you choose for the spectrogram. From the magnitude spectrum, identify the frequency band where white noise is present. Then, examine the spectrogram. Can you find more sources of noise in the spectrogram? Why does the magnitude spectrum fail to capture this? Use **firpm** to create a filter to remove the noise observed from the magnitude response. Plot the magnitude spectrum and spectrogram of the filtered signal using the same parameters as before. Additionally, use **soundsc** to listen to the filtered signal. Comment on the effectiveness of the filter.

7 Fast Convolution

7.1 Important (Maybe Interesting) History

It is well known that Gauss developed a fast algorithm for the DFT back in 1805 when he was interpolating orbits of asteroids. The fast DFT, or the Fast Fourier Transform, was developed by James Cooley of IBM and John Tukey of Princeton. The motivation of the FFT came from the days of the Cold War, when President Kennedy's Advisory committee wanted to monitor Soviet Nuclear experiments by planting seismic sensors in the ground in the USSR. The raw computational ability back in the 60s could not handle such spectral computations. As a result, Richard Garwin, a staff member of IBM, told Cooley to work with Tukey on a new algorithm to do fast spectral computation. The two published the algorithm in 1965. The FFT has found its place in many applications from CDMA to Shazam.

7.2 Linear Convolution - Slow

Okay, you know convolution is calculated by :

$$y[n] = x[n] * h[n] = \sum_{k=0}^{N-1} x[k]h[n-k] \tag{6}$$

Tom compute this expression would require either a for loop or a matrix operation. We'll consider the case of convolution as a matrix operation. Suppose $x[n]$ is a N length sequence and $h[n]$ is a M length sequence. We can define a matrix C, that is of dimensions $M+N-1 \times N$ in row and length.This is such that $y = Cx$. Here C is formed from $h[n]$. This matrix C takes the form a Toeplitz matrix. Suppose, we have a sequence $x[n] = 1, 2, 3, 4$ and we have $h[n] = 1, 2$. We have that $y = Cx$:

$$\begin{bmatrix} 1 \\ 4 \\ 7 \\ 10 \\ 8 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

You may notice that for large length signals that the computation and storage requirements become much larger.

7.3 Linear Convolution - Fast

We all know that convolution is equivalent to a multiplication in frequency with added step of an inverse Fourier Transform. For discrete sequences, this is true in the sense of circular convolution. Circular Convolution Theorem holds that :

$$y[n] = (\hat{x}[n] * \hat{h}[n])_N = DFT^{-1}(DFT(\hat{x}[n])DFT(\hat{h}[n])) \tag{8}$$

There are some caveats with this method. $x[n]$ and $h[n]$ must be zero-padded to the proper length for the circular convolution to be the same as the linear convolution. Your textbook author, Dmitris Manolakis, explains this theorem quite well; I will not detail the proof here. If $x[n]$ is a N length sequence and $h[n]$ is a M length sequence, then both sequences must be zero-padded to $N+M-1$. The zero padded version are $\hat{x}[n]$ and $\hat{h}[n]$ respectively. Now, we'll do some experiments with slow and fast computation.

Report Item 7:

1.) Load sig1.mat. The variable sig should be stored in your work space. Take out the first $N = 50$ samples from sig and store it in a variable called x.
2.) Define a filter $h[n]$ of length 24, with entries equal to $\frac{1}{24}$
3.) Zero Pad the $h[n]$ and $x[n]$ to $N+M-1$. Store the zero-padded sequences into $hz[n]$ and $xz[n]$.
4.) Perform the FFT based convolution in equation (8). Time this operation using 'tic' and 'toc'
5.) Form the convolution matrix C from $h[n]$ and the length N. Note that the $h[n]$ is not zero padded. You may use the function convmtx. Read the documentation if you'd like to do this.
6.) Apply the convolution matrix C on $x[n]$ to obtain $y[n]$. Does this match up with the result in step 4? In addition, time this matrix operation using 'tic' and 'toc', this should include the time it takes to form the convolution matrix.
7.) Repeat steps $1-6$ with $N = 100, 500, 1000, 10000$. Comment on the run time.

The revolution of the Fast Fourier Transform (FFT) allowed mankind to perform convolution really quickly, allowing people to filter information quick in addition to perform spectral analysis efficiently. In this section we will experimentally compare the performance of the Overlap Save method.

$$y[n] = x[n] * h[n] = \sum_{k=0}^{N-1} x[k]h[n-k] \tag{9}$$

7.4 Overlap Save - For interested readers

We all know that convolution is equivalent to a multiplication in frequency with added step of an inverse Fourier Transform. For discrete sequences, this is true in the sense of circular convolution. Circular Convoltuion Theorem holds that :

$$y[n] = (x[n] * h[n])_N = DFT^{-1}(DFT(x[n])DFT(y[n])) \tag{10}$$

In practice, we are only able to access a fixed length FFT (usualy a power of 2 size). Overlap-Save is an FFT-based convolution algorithm that operates on this constraint. The method that we will describe essentially breaks down the convolution to several L-length FFTs and will use equation (10) to compute segments of the convolution. Partitions of these individual convolution are continually stored in some resulting vector until all L-length partitions have been computed. The theory behind this is best explained by wikipedia in the following link: `https://en.wikipedia.org/wiki/Overlap-save_method`

The pseudocode for this algorithm is below:

Algorithm 1 Overlap Save

procedure OVERLAP SAVE
  $h = FIRFILTER$
  $M = length(h)$
  $overlap = M - 1$
  $N = POWEROF2$
  $stepsize = N - overlap$
  $H = DFT(h, N)$
  $position = 0$
While position+$N <=$ *length(x)*:
  $yt = IDFT(DFT(x(1 + position : N + position), N) * H, N)$
  $y(1 + position : stepsize + position) = yt(M : N)$
  $position = position + stepsize$